

# Towards the Optimization of Power and Bandwidth Consumption in Mobile-Cloud Hybrid Applications

Aamir Akbar

Aston Lab for Intelligent Collectives Engineering (ALICE)  
School of Engineering and Applied Science  
Aston University, Birmingham, UK  
Website: <https://cs.aston.ac.uk/~akbara2/>

Peter R. Lewis

Aston Lab for Intelligent Collectives Engineering (ALICE)  
School of Engineering and Applied Science  
Aston University, Birmingham, UK  
Website: <http://prlewis.com/>

**Abstract**—Mobile devices can now support a wide range of applications, many of which demand high computational power. Backed by the virtually unbounded resources of cloud computing, today's mobile-cloud (MC) computing can meet the demands of even the most computationally and resource intensive applications. However, many existing MC hybrid applications are inefficient in terms of achieving objectives like minimizing battery power consumption and network bandwidth usage, which form a tradeoff. To counter this problem we propose a technique that: 1) measures, at run time, how well the MC application meets these two objectives; and 2) allows arbitrary configurations to be applied to the MC application in order to optimize the efficiency tradeoff. Our experimental evaluation considers two MC hybrid applications. We modularized them first, based on computationally-intensive tasks, and then executed them using a simple MC framework while measuring the power and bandwidth consumption at run-time. Analysis of results shows that efficient configurations of the apps can be obtained in terms of minimizing the two objectives. However, there remain challenges such as scalability and automation of the process, which we discuss.

## I. INTRODUCTION

In recent years, there has been a growing interest in binding cloud resources to low-power devices such as smartphones in order to provide PC-like functionality to mobile users [1]. Integrating cloud computing technology [2] and mobile devices makes the latter virtually limitless in terms of processing power, energy and storage space. This interdisciplinary domain is called Mobile-Cloud Computing (MCC) [3]. A mobile device, i.e. a smartphone, can either be connected via WiFi or a 3G/4G data network. In both cases sending/receiving data packets over the network uses bandwidth and therefore consumes device battery power.

Mobile-Cloud (MC) hybrid applications are becoming ever more frequently relied upon. However, they have an efficiency tradeoff between energy consumption and network bandwidth usage. This exists because, 1) performing computationally intensive tasks on mobile devices can be inefficient in terms of battery power consumption, 2) sending data to and receiving from the cloud can also be inefficient in terms of bandwidth usage cost and battery power consumption by the transmitting chip (i.e. WIFI) inside the mobile device. We consider this as a multi-objective optimization problem.

In multi-objective optimization there are a number of objectives to be optimized simultaneously, and typically the objectives are conflicting with each other. Thus there exists a tradeoff between objectives, which leads to the idea of a set of Pareto-optimal solutions [4], those that are not dominated by any possible other solution in the solution space.

In this paper we present 1) the use of search-based techniques for multi-objective optimization, to optimize power consumption and bandwidth usage of MC hybrid apps, 2) a workflow that uses this to find energy efficient configurations for MC hybrid apps, 3) key challenges, including automatically partitioning MC hybrid apps, applying search-based algorithms for multi-objective optimization and efficient communication in between a mobile device and a cloud server-machine.

The rest of this paper is organized as follows. Section II discusses the related work. Section III contains the problem description. Section IV introduces the case studies and configuration sets. Section V explains the workflow used to find the optimal configurations from the sets. Section VI explains the experiments conducted and their results. Section VII explains the challenges and directions for future work. Section VIII concludes the paper.

## II. RELATED WORK

The design of MC hybrid applications is normally based on achieving one particular objective such as minimising energy use, maximising application performance, or minimising bandwidth usage. Achieving two or more objectives at the same time might not be possible, for example, minimizing bandwidth usage may prevent the objective of minimizing power consumption. Based on their objectives, Khan et al. [5] classified MCC application frameworks into four categories: 1) performance-based, 2) energy-based, 3) constraint-based, and 4) multi-objective-based frameworks. Using such a framework, mobile applications can be customized and integrated with the cloud, for example through a mediator (aka middleware) that provides an abstraction, and controls every aspect of the communication in between the mobile and the cloud server. Two criteria for middleware are normally considered: 1) delegating a mobile task by invoking a cloud service from the handset, 2) partitioning the application source code into

modules (e.g. methods, classes) and analyzing a priori (at development time) or a posteriori (at runtime) so that the most computationally intensive modules at the code level can be identified and offloaded for remote processing [6] [7].

#### A. Mobile-Cloud Application Frameworks

MC frameworks are designed to foster the development of hybrid apps by providing guidelines and APIs to developers. In the following sections we will discuss some common frameworks.

1) *MAUI*: MAUI [8] is an MC hybrid app framework that combines the benefits of two approaches: it maximizes the potential for energy savings through fine-grained code offloading, while minimizing the changes required to applications. MAUI enforces method-level partitioning of mobile applications, which is done using method annotating (with `@Remoteable`) at development time. At runtime, the remoteable methods are identified using the Microsoft .Net Reflection API. The MAUI solver decides whether a method can be offloaded to the cloud or kept executing on the mobile device.

2) *CloneCloud*: CloneCloud [9] offloads computationally intensive parts of the application code to a device clone operating in the cloud, to enhance application performance. Unlike MAUI, CloneCloud operates at thread granularity. It uses a combination of static analysis and dynamic profiling to partition applications automatically at a fine granularity, while optimizing execution time and energy use for a target computation environment. As there are no annotations in the source code, the partitioning mechanism is used off-line, which aims to pick which parts of an application's execution to retain on the mobile device and which to migrate to the cloud. The partitioning mechanism may be run multiple times for different execution conditions and objective functions, resulting in a database of partitions. At runtime, the distributed execution mechanism picks a partition from the database and implements it via a small and fast set of modifications of the executable before invocation.

3) *ThinkAir*: ThinkAir [10] addresses two key issues that were lacking in MAUI and CloneCloud: parallelism of method execution using multiple virtual machine images on the cloud, and adaptation of online method-level offloading. Like MAUI, ThinkAir operates at method granularity where methods are annotated by developers during development. Unlike MAUI, ThinkAir is Java-based and uses its Reflection API to remote-execute the code. The calling object is sent to the server endpoint in the cloud; the mobile endpoint then waits for results, and any modified local state, to be returned.

4) *Mobile Cloud Middleware*: Flores et al. [11] proposed a generic middleware framework for handling process-intensive hybrid cloud services from a mobile device. The objective of MCM is to handle the interoperability issues of different APIs and services used by different cloud vendors. Using a middleware can provide support in both task delegation and code offloading from mobile to cloud. A middleware can also be useful for buffering the intermediate data to be used by either a mobile or cloud.

#### B. Search-based Multi-objective Optimization

A search problem is defined as finding an optimal or near-optimal solution(s) in a space of candidate solutions, guided by a fitness function that distinguishes between better and worse solutions [12]. Search-based Software Module Clustering (SMC) techniques have been used for comprehending big software systems, by formulating the basic software's design metrics of high cohesion and low coupling into a fitness function [13]. Praditwong et al. [14] proposed a search-based approach to SMC with multi-objective fitness function. Unlike single objective formulation, they considered the objectives, cohesion and coupling, separately.

Similar to multi-objective optimization in SMC, we minimize both bandwidth and power consumption to find efficient configurations for MC hybrid applications. We developed a simple framework based on class-level tasks delegation to cloud and using a middleware to facilitate the system development on a network. We choose *Firebase*<sup>1</sup> as a middleware, which is a real-time, event-driven database service provided by Google. Like CloneCloud, we used offline execution of the search algorithm to find Pareto-optimal configurations. During the runtime of an application we measured the battery power consumption and network bandwidth usage to find the efficiency of a configuration.

### III. PROBLEM SETTING

Mobile devices can access the cloud either through WIFI or a 3G/4G data network. From a connection point of view, the decision to execute computationally-intensive tasks of MC hybrid applications on the cloud can be affected by the network bandwidth. For instance, executing the code on the cloud at runtime can demand high bandwidth when sending a large amount of data. From a mobile device point of view, executing computationally-intensive tasks on device can demand high battery power consumption. To attain an optimal configuration of MC hybrid applications while not affecting the overall performance of the application is considered a challenging area in MCC [15]. We define a *configuration* as a valid mapping of all the distinct tasks of an MC hybrid application to mobile and cloud server endpoints, indicating where they are to be executed.

Assume that an MC hybrid application is composed of a set of collaborative code units called modules (e.g. classes in Java), which can be executed on either mobile-device or cloud-server endpoint. In general, when designing an MC hybrid application we are faced with a decision about which module should be placed on the mobile device and which on the cloud. As there exists an efficiency tradeoff between the network bandwidth usage and battery power consumption, a set of optimal configurations of MC hybrid applications can be achieved. Power consumption is at maximum when executing the computationally-intensive modules on the mobile device. Network bandwidth usage is at maximum when an application rely on cloud by sending and receiving large amount of

<sup>1</sup><https://firebase.google.com/>

data. Measuring network bandwidth usage and mobile device battery power consumption, we might obtain Pareto-optimal configurations. Therefore, we consider MC hybrid application development as a multi-objective problem by optimizing these two objectives.

- 1) Minimize power consumption: number of Joules consumed by the application on a mobile device, during one execution.
- 2) Minimize network bandwidth: data sent and received between a mobile device and a cloud server during one execution (MBs).

#### IV. CASE STUDIES

Two different Android applications were considered as test problems: a prototype app we created called *ImageEffects* and an open source *Chess* game<sup>2</sup>.

*ImageEffects* has some built-in image-filters to apply to an image, taken by using device camera inside the app. The app can also be connected to a Dropbox account to upload the filtered image. The mobile battery power is consumed when performing computationally-intensive tasks i.e. image processing. Also, it uses network bandwidth during uploading/downloading images to/from Dropbox. The Chess game has a board and pieces on it. When played against computer (User-PC mode), it uses a built-in Artificial Intelligent (AI) search algorithm to find the best and a valid move for the computer's pieces on the board. When the game playing level is increased the search algorithm will take more time to search for that move. The AI unit, therefore, is computationally-intensive which consumes battery power when executed on a mobile device. To execute across mobile and cloud platforms both of these applications need to be customized first according to the platform. The customization steps and creating their configuration sets are discussed below.

##### A. *ImageEffects* Application

*ImageEffects* consumes battery power while performing image processing tasks. There are five distinct tasks that can consume battery power. The first task is applying a filter to an image, which is to change each and every pixel of the original image. The second task is to generate a thumbnail of the filtered image, which is scaling down the image to a lower resolution. The third task is uploading the filtered image to the user's Dropbox account. Downloading the image back to the mobile device is then the fourth task. The fifth and last task includes two sub parts: 1) calculating hash codes of the original filtered image and the downloaded image from the Dropbox, 2) comparing the hash codes to verify the downloaded picture. This app is designed in such a way that each of these five tasks are performed in their respective modules (Java classes).

After the app is modularized the next step is to create its configuration set. A configuration is a binary string representation of where each module performs their tasks: mobile

or cloud. If a module is 0 it will be executed on a mobile device and if it is 1 then it will be executed on the cloud. For example, assuming a configuration 00101, the modules at 1<sup>th</sup>, 2<sup>nd</sup> and 4<sup>th</sup> locations represent the respective tasks which will be executed on the mobile device as their states are 0. Similarly the 3<sup>rd</sup> and 5<sup>th</sup> tasks will be performed on the cloud server as their states are indicated as 1. For 5 modules,  $n = 5$ , the cardinality of the configuration set  $S_i$  for *ImageEffects* is  $2^n = 32$ . The set  $S_i$  starting with 00000 where all the modules will execute on the mobile device and ending with 11111 where all the modules will execute on the cloud server.

##### B. *Chess* Game

The Chess game is composed of modules (Java classes) that form two main components of the game: 1) the User Interface (UI) component that will only be executed on a mobile device, and 2) the AI component that is composed of a search algorithm to find the best and a valid move of the pieces on the board. Executing the AI component, therefore, can be performed on cloud to minimize battery power consumption of the mobile device. One parameter of the AI is the game playing level, which determines the time duration that the search algorithm is allowed to find the best move. Increasing the game playing level will increase the time duration of search algorithm. So, with higher game levels more computation will be done, which will have direct effect on more power consumption.

A configuration string for the Chess game will represent the two computationally-intensive tasks we identified. We used alphanumeric representation of this string. The first element of the string can either be *M* or *C*, representing the first task (AI component) to be executed on either a mobile or cloud respectively. The second element of the string is for the second task which is represented with 5 different game playing levels; 1, 4, 7, 10 and 15. For example, assuming a configuration string *M4*, the AI unit will execute on a mobile with the game level set to be 4. For the 5 levels and executing AI component on 2 machines, the cardinality of the configuration set  $S_c$  for the Chess game is  $2 \times 5 = 10$ . The set starting with *M1* where the first task will be performed on the mobile device with game level 1 and ending with *C15* where the first task will be carried out on the cloud with game level 15.

#### V. WORKFLOW TO FIND OPTIMAL CONFIGURATIONS

To evaluate the possible configurations, the MC hybrid apps need to be executed several times. For this purpose, automation of the workflow is required. We used a pure Python based open source library called *AndroidViewClient*<sup>3</sup>. It provides higher level operations and the ability to obtain a tree of Android UI Views present at any given moment on the device or emulator screen and performs operations on it. Alternatively, Android's own library *MonkeyRunner* can also be used for the same purpose. We wrote a Python script, using the *AndroidViewClient*, that interacts with the MC hybrid

<sup>2</sup><https://github.com/jcarolus/android-chess>

<sup>3</sup><https://github.com/dtmilano/AndroidViewClient/wiki>

applications that will be installed on a mobile device. The mobile device must be connected to a PC on which this script will be executed. The script will launch the MC hybrid apps on the device automatically with a valid configuration from their respective sets and then will send touch events to interact with the GUI.

To find optimal configurations in the two sets,  $S_i$  and  $S_c$ , a search algorithm is required. Due the fact that both of these sets are not big in size, they can be searched exhaustively. Therefore, we implemented an exhaustive search inside the automation script. For a large configuration set, search algorithms like genetic algorithms could be used.

Algorithm 1 shows the automation script which begins with making a connection to the attached mobile device. The script then enters into a loop which iterates through all the configurations in  $S$  and applies each to the MC application. To measure the network bandwidth usage and power consumption on the device we created a separate monitor application that is executed on the device before running the MC hybrid app. The power consumption is recorded by using an open source tool, integrated in the Monitor application, called PowerTutor<sup>4</sup>. Its profiler monitors the total power consumption of each of the components (WIFI, CPU and LCD) the MC hybrid app uses. Also, it monitors the network bandwidth usage of the app for the current configuration.

Inside the inner loop of the script, the MC hybrid application is checked repeatedly. When it is found stopped/destroyed, a touch event stopping the monitor app is issued. At the end of the script, the *count* variable is incremented and this process is repeated until all the configurations in  $S$  are tested. Receiving measurements from the monitor application is also done automatically. For this we created a log receiving script that gets measurements from the monitor app during each iteration of the outer loop. The monitor writes the measurements using Android log library. The log is then parsed and all the measured data is stored in a dedicated MYSQL database on the PC.

## VI. EXPERIMENTS AND RESULTS

The experiments were conducted on two different Android-based smartphones: 1) Samsung Galaxy S5 (a high end smartphone), and 2) ZTE Skate (a low end smartphone). Both the MC hybrid apps, ImageEffects and Chess game, along with the Monitor app were installed on the phones. The server counterparts of the MC hybrid apps were installed on the cloud server endpoint. At the Firebase middleware, all the parent-child keys were set for messaging and buffers. The automation script and the script for reading log messages from the phones were both started on a PC.

The workflow, discussed in section V, was repeated 30 times for each of the MC hybrid apps and for both phone separately. As a result, 30 samples of each configuration from the sets  $S_i$  and  $S_c$  were collected. Each sample has data points that include total network bandwidth usage (MB) and

---

### Algorithm 1 Automation Script with Exhaustive Search

---

```

1: device  $\leftarrow$  connectToDevice()
2: if device == null then
3:   return
4: end if
5:  $S \leftarrow$  ConfigurationSet
6: count  $\leftarrow$  1
7: while count  $\leq n$  do
8:   device.shell(Monitor App,  $S[\textit{count}]$ )
9:   device.shell(MC App,  $S[\textit{count}]$ )
10:  while true do
11:    app  $\leftarrow$  device.shell(ps | grip  $\alpha$ )
12:    if app == null then
13:      break
14:    end if
15:  end while
16:  device.touch(x,y) ▷ Stop the Monitor app
17:  count  $\leftarrow$  count + 1
18: end while

```

---

total power consumption (Joules). After the samples were collected, the mean and standard deviation (SD) of power (battery power consumption) and data (network bandwidth usage) were calculated.

#### A. Results from ImageEffects

In case of the ImageEffects, we obtained 3 different Pareto-optimal configurations for each of the phones as shown in table I. Figures 1 and 2 show the efficiency of all the configurations for Samsung Galaxy S5 and ZTE Skate respectively. Only the Pareto-optimal configuration and the non-hybrid configurations (00000 and 11111) are labeled in these graphs.

#### B. Results from the Chess Game

The measured efficiency of all the configurations of the Chess game from its set  $S_c$  is shown in table II. These configurations are plotted on graphs shown in figures 3 and 4 for Samsung Galaxy S5 and ZTE Skate respectively. It is clear from these graphs that executing the AI unit on cloud is energy efficient for high game levels.

## VII. CHALLENGES AND FUTURE WORK

In order to validate the idea of optimizing bandwidth and power consumption in MC hybrid apps, we reported on a small scale manually carried out feasibility study and identify some key challenges. We will now discuss them along with the future directions.

- **Automation:** Identifying the computationally-intensive tasks and then partitioning the apps were done manually during development. For large applications (in terms of modules/tasks), automatic partitioning is required, which could be done a priori (using code annotations) or a posteriori (using bytecode analysis).
- **Scalability:** Exhaustively searching a large configuration set (more than 10 modules) will take more time. Instead,

<sup>4</sup><https://github.com/msg555/PowerTutor>

TABLE I  
PARETO-OPTIMAL CONFIGURATIONS OF IMAGEEFFECTS

Mobile Device	Configuration	Battery Power Consumption (J)		Network Bandwidth Usage (MB)	
		Mean	Standard Deviation	Mean	Standard Deviation
Samsung Galaxy S5	00000	132.2	17.4612	4.7577	0.0186
	00010	153.8	23.9908	2.998	0.0108
	10100	85.6667	15.6617	5.198	0.0166
ZTE Skate	00110	275.1	0.21	25.3763	0
	00010	1642.8333	815.6709	0.18	0
	10110	220.4	0.32	27.3186	0

TABLE II  
CONFIGURATIONS OF CHESS GAME

Configuration	Samsung S5				ZTE Skate			
	Mean Power	SD Power	Mean Data	SD Data	Mean Power	SD Power	Mean Data	SD Data
M1	39.5	10.9446	0	0	306.4667	17.2525	0	0
M4	225.1333	42.2632	0	0	562.9333	120.2192	0	0
M7	645	69.3792	0	0	759.3667	74.2574	0	0
M10	1152.6667	116.3178	0	0	1947.5667	115.2535	0	0
M15	1654.9	124.3156	0	0	4255.3	617.1204	0	0
C1	39.8667	8.7206	0.03	0	373.8	19.6289	0.02	0
C4	80.6	11.5833	0.03	0	710.6667	57.4179	0.02	0
C7	188.7	17.8795	0.03	0	1571.9333	428.913	0.02	0
C10	268	33.5102	0.03	0	2074.9667	270.6117	0.02	0
C15	433.9	39.5890	0.03	0	3380.8333	336.0201	0.02	0

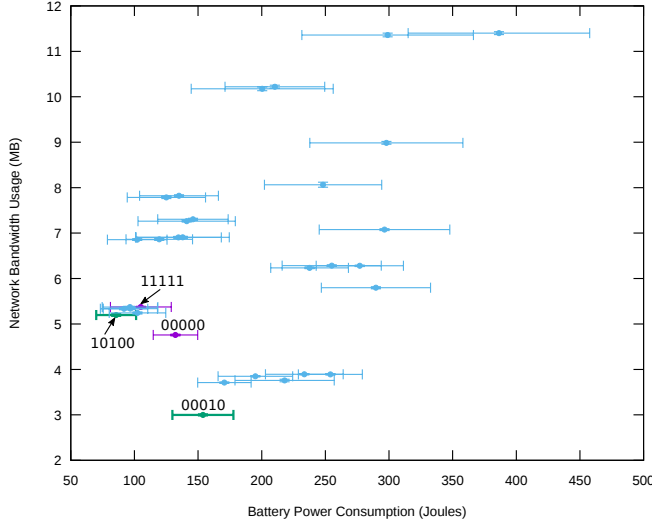


Fig. 1. ImageEffects on Samsung S5.

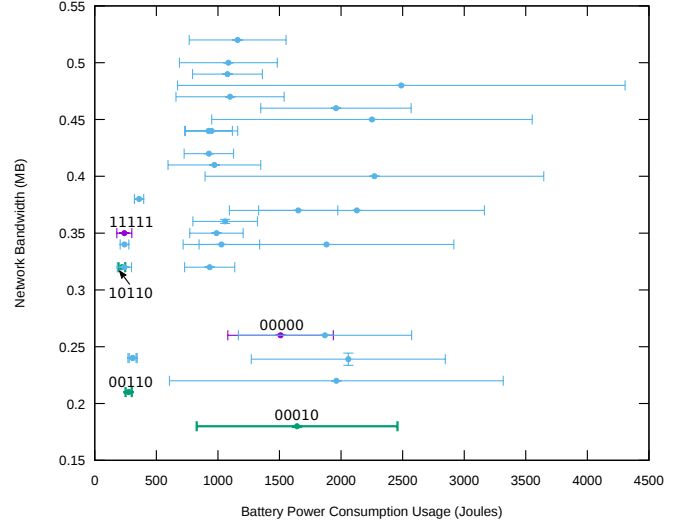


Fig. 2. ImageEffects on ZTE Skate.

using Evolutionary Algorithms such as GAs is likely to take less time to find a good solution.

- Efficient communication: Firebase works in an event-driven fashion which calls back to the thread that starts its on-event handler. Using Firebase or other such kind of systems as middleware might not be a good choice. It is due to the fact that if the waiting thread is the UI/main thread to get the results from the cloud, the callback from the Firebase handler will be blocked due to the inherent characteristic of the Android systems. Instead, socket based middleware could be used, which works in a suspend-migrate-receive-resume fashion.

## VIII. CONCLUSION

In this paper, we presented a technique for developing efficient MC hybrid apps by optimizing network bandwidth usage and battery power consumption. Using this technique, an application's source code is modularized on the bases of identifying such tasks that consumes battery power of a mobile device. Applying a configuration that represents those modules/tasks, an MC hybrid application is executed using a middleware based task delegation from mobile to cloud. The battery power consumption and network bandwidth usage is then measured at runtime of the application. We

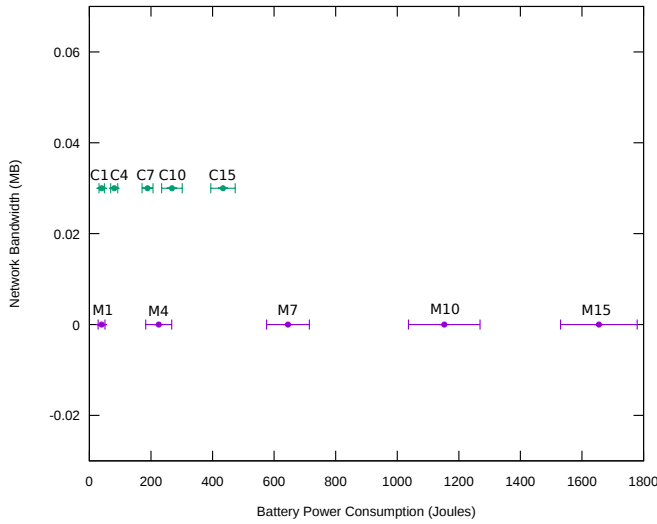


Fig. 3. Chess game on Samsung S5.

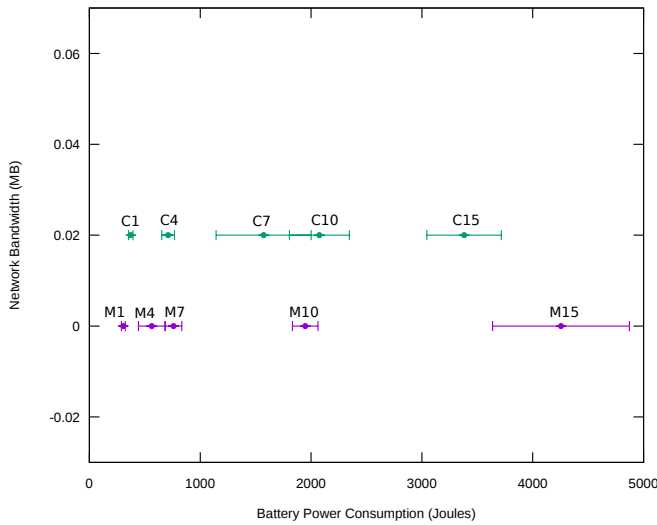


Fig. 4. Chess game on ZTE Skate.

conducted experiments by applying this technique on two different Android based applications and on two different smart phones. The workflow to evaluate the configurations was presented.

The experimental results obtained indicate that one of the two MC hybrid applications, ImageEffects, produced Pareto-optimal configurations for both phones. Such configurations are efficient in terms of optimizing both of the two objectives. The second application, Chess, also produced efficient configurations. For a high-end mobile phone like Samsung Galaxy S5, executing the game AI on the cloud consumed less battery power, at the cost of using more network bandwidth. For a low-end phone like ZTE Skate, the results were in favor of executing the game AI on the phone for lower game levels. The results from both apps together also indicate that efficient configurations vary between different mobile devices.

At the end we presented the key challenges with future directions. It is our hope that this technique will provide researchers and developers insights to make use of mobile-cloud computing technologies and developing MC hybrid applications.

## REFERENCES

- [1] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang, "The case for cyber foraging," in *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, ser. EW 10. New York, NY, USA: ACM, 2002, pp. 87–92.
- [2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.
- [3] D. Huang *et al.*, "Mobile cloud computing," *IEEE COMSOC Multimedia Communications Technical Committee (MMTC) E-Letter*, vol. 6, no. 10, pp. 27–31, 2011.
- [4] K. Deb, "Multi-objective optimisation using evolutionary algorithms: an introduction," in *Multi-objective evolutionary optimisation for product design and manufacturing*. Springer, 2011, pp. 3–34.
- [5] A. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 1, pp. 393–413, 2014.
- [6] X. Gu, A. Messer, I. Greenberg, D. Milojicic, and K. Nahrstedt, "Adaptive offloading for pervasive computing," *IEEE Pervasive Computing*, vol. 3, no. 3, pp. 66–73, Jul. 2004.
- [7] Z. Li, C. Wang, and R. Xu, "Computation offloading to save energy on handheld devices: A partition scheme," in *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, ser. CASES '01. New York, NY, USA: ACM, 2001, pp. 238–246.
- [8] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 49–62.
- [9] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 301–314.
- [10] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Infocom, 2012 Proceedings IEEE*. IEEE, 2012, pp. 945–953.
- [11] H. Flores, S. N. Srirama, and C. Paniagua, "A generic middleware framework for handling process intensive hybrid cloud services from mobiles," in *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia*, ser. MoMM '11. New York, NY, USA: ACM, 2011, pp. 87–94.
- [12] M. Harman, P. McMinn, J. T. de Souza, and S. Yoo, "Empirical software engineering and verification," B. Meyer and M. Nordio, Eds. Berlin, Heidelberg: Springer-Verlag, 2012, ch. Search Based Software Engineering: Techniques, Taxonomy, Tutorial, pp. 1–59.
- [13] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner, "Using automatic clustering to produce high-level system organizations of source code," in *Proceedings of the 6th International Workshop on Program Comprehension*, ser. IWPC '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 45–.
- [14] K. Praditwong, M. Harman, and X. Yao, "Software module clustering as a multi-objective search problem," *IEEE Trans. Softw. Eng.*, vol. 37, no. 2, pp. 264–282, Mar. 2011.
- [15] E. Ahmed, A. Gani, M. Sookhak, S. H. A. Hamid, and F. Xia, "Application optimization in mobile cloud computing," *J. Netw. Comput. Appl.*, vol. 52, no. C, pp. 52–68, Jun. 2015.